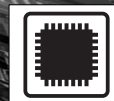
初歩からのHDLテストベンチ

第3回　絶対遅延とソフトウェア風テストベンチの文法

安岡貴志

デバイスの記事

ビギナーズ

HDLで回路を記述できるようになったばかりで、これからテストベンチを書こうとしてる方を対象とした連載の第3回である．始めに絶対時間で遅延を記述する方法を解説し，サンプル回路の検証事例をもとに，遅延の効率的な記述法を説明する．（筆者）

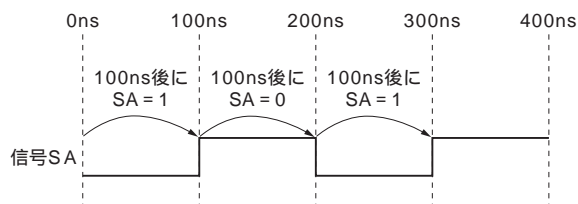
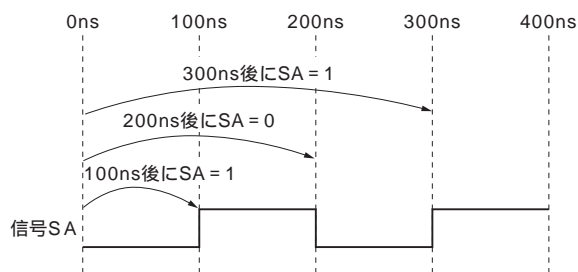
前回（本誌2007年8月号，pp.116-124）まではテスト入力（検証対象回路の入力ポートに与える信号）を作るのに相対遅延を使っていました．今回は絶対遅延による表現を解説します．また，第1回（本誌2007年5月号，pp.70-79の特集1 第4章編集部注）で，テストベンチではHDLのすべての文法を，それこそC言語によるソフトウェア・プログラムのように使えると言いました．今回はその中からfor文によるループ表現を説明します．

（a）遅延を前の信号の変化からの相対遅延で表現

（b）遅延をシミュレーション開始時からの絶対遅延で表現

図1　遅延の表現
前の信号の変化からの相対関係で表現する相対遅延と，常に開始時点からの時間で表現する絶対遅延がある．

1．相対遅延と絶対遅延

相対遅延とは，前の信号の変化から何nsで代入を実行するというように，遅延を相対関係で表現する手法です［**図1（a）**］．前回までのテストベンチはすべてこの手法で書いてきました．これに対して絶対遅延とは，シミュレーション開始時点（シミュレーション時間0）から何nsで代入するというような，絶対時間で表現する手法です［**図1（b）**］．

ここでは，第1回で示した**図2**の仕様をもつ検証対象の回路に対して，**図3（a）**のテストベンチから，**図3（b）**の信号SAと信号SBのようなテスト入力を与える記述法を，相対遅延，絶対遅延を使って解説します．

編集部注：本連載の第1回は，本誌2007年5月号，pp.70-79の特集1 第4章「テストベンチの書き方を身に付ける」として掲載した．

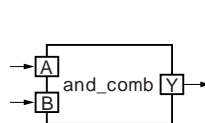
A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

（a）ブロック図　（b）真理値表

図2　検査対象の回路
回路の名前は，and-combである．1ビットの入力ポートA，Bと1ビットの出力ポートYを持つ．記憶素子（フリップフロップなど）を含まない組み合わせ回路である．

KeyWord　テストベンチ，テスト入力，絶対時間，相対時間，fork，wait，after，assert文，エンコーダ，for文

Verilog HDL

リスト1(a)は，第1回で示したテスト入力の記述例です．ここで記述された信号SA，SBの変化のタイミングは**図3(b)**と同じです[注1]．

リスト1(b)は，**リスト1(a)**の記述を1行当たり一つの式だけの記述に直したものです．式の実行タイミングは，式の間の相対遅延で記述されています．**図4**のように，begin～endの間に記述された式は上から順番に実行され，その間の遅延は累積されます（式2は必ず式1の後に実行され，式3は必ず式2の後に実行される）．

図5はfork～joinという文法の書式を表しています．fork～joinは，begin～endと違い，式の間で遅延は累積しません．それぞれの式がシミュレーション開始時から独自の遅延値に従って並行に実行されます．つまり，各式の遅延値は絶対遅延となります．

リスト2は，**リスト1**とまったく同じ変化タイミングで信号SAとSBのテスト入力を記述したものです．各代入式の#に続く遅延値は絶対遅延となり，式の間で累積しません．

VHDL

リスト3(a)は，第1回で示したテスト入力の記述例です．ここで記述された信号SA，SBの変化のタイミングは，**図3**と同じです．

リスト3(b)は，**リスト3(a)**の記述を1行当たり一つの式だけの記述に直したものです．式の実行タイミングは，wait文ごとに累積する相対遅延で記述されています．**図6**のように，wait文を使った記述では，式2は式1実行後，間のwait文で指定した遅延経過後に実行され，式3は式2実行

注1　本稿のVerilog HDLのすべての解説では，シミュレータのシミュレーション時間の単位が，nsに設定されているものとして解説しています．

(a) ブロック図

(b) タイミング・チャート

図3　AND回路の検証

(a)のようなテストベンチから，(b)のテスト入力を与える．

図4
Verilog HDLによる相対遅延の書式

begin～endの間の式は，上から順に実行され，遅延が累積する．

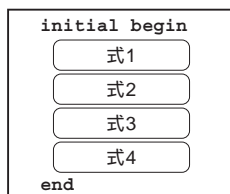



図5
Verilog HDLによる絶対遅延の書式

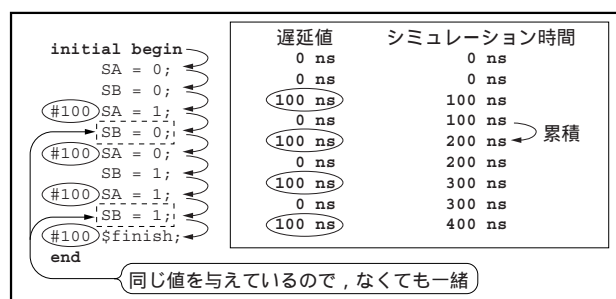
fork～joinの間の式は，シミュレーション開始時点からの独自の遅延値に従って並行して実行され，式の間で遅延は累積しない．

リスト1　Verilog HDLによるAND回路のテストベンチ

```
initial begin
        SA = 0; SB = 0;
#100 SA = 1; SB = 0;
#100 SA = 0; SB = 1;
#100 SA = 1; SB = 1;
#100 $finish;
end
```

(a) 記述例1

```
initial begin
        SA = 0;
        SB = 0;
#100 SA = 1;
        SB = 0;
#100 SA = 0;
        SB = 1;
#100 SA = 1;
        SB = 1;
#100 $finish;
end
```

遅延値	シミュレーション時間
0 ns	0 ns
0 ns	0 ns
100 ns	100 ns
0 ns	100 ns
100 ns	200 ns
0 ns	200 ns
100 ns	300 ns
0 ns	300 ns
100 ns	400 ns

(b) 記述例2

リスト2　Verilog HDLによる絶対遅延を使ったテストベンチ

```
initial fork
        SA = 0;
   #100 SA = 1;
   #200 SA = 0;
   #300 SA = 1;
        SB = 0;
   #200 SB = 1;
   #400 $finish;
join
```

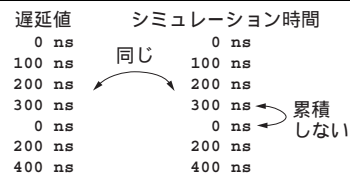
遅延値	シミュレーション時間
0 ns	0 ns
100 ns	100 ns
200 ns	200 ns
300 ns	300 ns
0 ns	0 ns
200 ns	200 ns
400 ns	400 ns

後，間のwait文で指定した遅延経過後に実行されます．

図7はafterを使った遅延の記述です．信号への代入は，afterを使って指定した値だけ遅らせられます．一つの信号に対する時間をずらした値の代入は，“，”で区切って一つの代入式の中に書き並べられます．**図7**の値0と値3の後ろにはafterが付いていないので，シミュレーション時間0で信号1，信号2に代入されます．値1，値2，値4もそれぞれシミュレーション時間0から遅延値1，遅延値2，遅延値3の経過後に代入されます．信号1のすべての代入が終わってから，信号2の代入が行われるわけではありません．

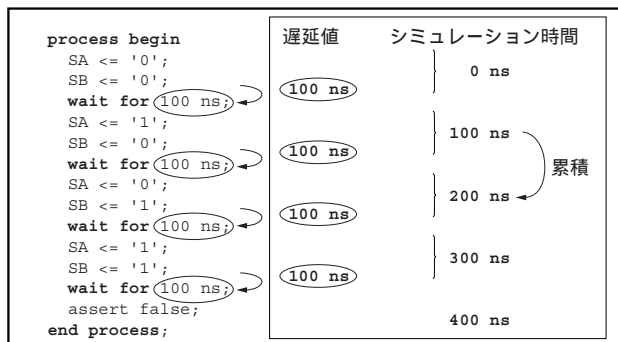
リスト4は，**リスト3**とまったく同じ変化タイミングで信号SAとSBのテスト入力を記述したものです．afterを使った代入では，遅延は絶対遅延となり，代入する値や式間で累積しません．ただし，assert文の実行はafterを使って遅らせられないので，wait文による相対遅延を使っています．

Verilog HDL　VHDL

絶対遅延を用いてテスト入力を書くと，一つの信号への代入をまとめて書いたり，ほかの式の遅延を考慮したりする必要がなくなるなど，いくつかの点でメリットがあります．テストベンチではどちらで記述しても構いません．好きな方を使えます．

リスト3　VHDLによるテストベンチ

```
process begin
                  SA <= '0'; SB <= '0';
  wait for 100 ns; SA <= '1'; SB <= '0';
  wait for 100 ns; SA <= '0'; SB <= '1';
  wait for 100 ns; SA <= '1'; SB <= '1';
  wait for 100 ns; assert false;
end process;
```

(**a**) 記述例1

(**b**) 記述例2

2．ソフトウェア風にテスト入力を記述する

検証対象となる回路記述では，最終的にASIC（Application Specific Integrated Circut）やFPGA（Field Programmable Gate Array）になるため，フリップフロップや実物の回路を意識して記述しなければなりません．このため，HDLの文法の中でも一部の文法しか使えませんでした．しかし，テストベンチに関しては，シミュレータの中でだけ機能すればよいので，HDLのすべての文法が使えます．

連載第1回と第2回で紹介したテストベンチでは，テスト入力は単にシミュレーション開始時点からの信号の変化をすべて書き並べる手法をとっていました．確かにこれでも検証は行えますが，テスト入力の値の変化が単調であったり，一定のパターン（法則）があるのであれば，ループ文を使うことにより，より簡単に記述できます．

ここでは，**図8**のような回路を検証します．この回路は入力ポートDINに入ってくる8ビットの信号の中で，何ビット目の信号が1になっているかを，4ビットで出力ポー

図6
VHDLによる相対遅延の書式
wait文の後の式は，wait文の前の式の実行後からwait文で指定した時間の経過後に実行される．遅延は累積する．

図7
VHDLによる絶対遅延の書式
afterを使った代入では，遅延は累積しない．

```
process begin
  信号1 <= 値0,
           値1 after 遅延値1,
           値2 after 遅延値2;
  信号2 <= 値3,
           値4 after 遅延値3;
end process;
```

リスト4　VHDLによる絶対遅延を使ったテストベンチ

トDOUTから出力する組み合わせ回路です．入力ポートDINに接続された信号のうち，複数のビットが1になっていた場合には，何を出してもよいという仕様です．

この回路を検証するために，**図9(a)**のDINのような信号を回路に与えて，シミュレーションします．回路が正しければ，DOUTのような出力があるはずです．

なお，エンコーダとは一般に，入力データを変換して出力する回路で，入力データのビット幅が出力データのビット幅よりも多いものをいいます．

テストベンチの構成を**図9(b)**に示します．検証対象の回路のポート名に対して，テストベンチ内の信号には先頭にSを付けて，見分けられるようにしてあります．

DIN	DOUT
10000000	111
01000000	110
00100000	101
00010000	100
00001000	011
00000100	010
00000010	001
00000001	000
上記以外	don't care

(a) ブロック図　(b) 真理値表

図8　エンコーダ回路のブロック図

入力ポートDINに入ってくる8ビットの信号の中で，何ビット目の信号が1になっているかを，4ビットの出力ポートDOUTから出力する組み合わせ回路である．

Verilog HDL

図9のような信号を与えるとき，連載第1回で解説した方法で記述すると**リスト5**のようになります．`initial`文の中を見ると，非常に似た代入を繰り返しています．

リスト5　Verilog HDLによるエンコーダ回路のテストベンチ

```
module encode_tb;
reg  [7:0] SDIN;
wire [2:0] SDOUT;
parameter STEP = 100;

encode encode(.DIN(SDIN),.DOUT(SDOUT));

initial begin
  // DINの各ビットに順次1を印加
        SDIN = 8'b00000001;
  #STEP SDIN = 8'b00000010;
  #STEP SDIN = 8'b00000100;
  #STEP SDIN = 8'b00001000;
  #STEP SDIN = 8'b00010000;
  #STEP SDIN = 8'b00100000;
  #STEP SDIN = 8'b01000000;
  #STEP SDIN = 8'b10000000;
  #STEP $finish;
end

endmodule
```

(a) タイミング・チャート

(b) テストベンチの構成

図9
エンコーダ回路の検証

DINのような信号を回路に与えて，シミュレーションする．回路が正しければ，DOUTのような出力がある．DINの値は16進表記．

このような処理は，for 文を使うと簡単に書けます．for 文の書式と動作を**図10**に示します．**図10(a)**のステートメントとは，Verilog HDLの一つの文（“；”から“；”まで）のことです．複数の文を一つのステートメントにまとめるときは，begin～endで囲います．

リスト6は，for 文を使って**リスト5**とまったく同じタイミングのテスト入力を書いたものです．for 文による処理の様子を**図11**に示します．

リスト6では，integer 宣言がされています．integer 宣言は，regと同じレジスタ型の変数で，符号付き32ビットの整数の値をとり，initial 文やalways 文の中で値を代入することができます．integer 宣言は，回路記述ではまず使いませんが，テストベンチの中では頻繁に使います．

VHDL

図9のような信号を与えるとき，連載第1回で解説した方法で書くと，**リスト7**のようになります．process 文の中を見ると，非常に似た処理を繰り返しています．このような処理はfor 文を使うと簡単に書けます．

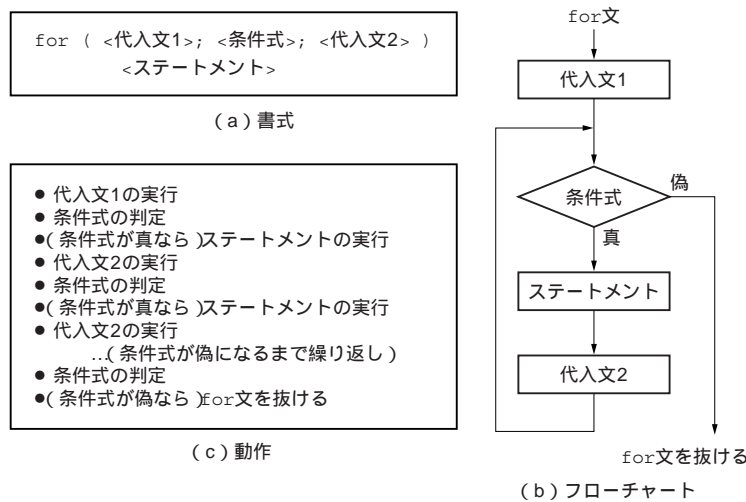



図10 Verilog HDLによるfor文の書式と動作
Verilog HDLのループ文である．

リスト6 Verilog HDLによるfor文を使ったテストベンチ

```
reg  [7:0] SDIN;
wire [2:0] SDOUT;
parameter STEP = 100;
integer    I;

encode encode(.DIN(SDIN),.DOUT(SDOUT));

initial begin
  // DINの各ビットに順次1を印加

  for( I=0; I<8; I=I+1 ) begin
    SDIN    = 8'h00;
    SDIN[I] = 1'b1;
    #STEP;
  end
  $finish;
end
```

（注記：integer宣言／代入文1／条件式／代入文2／for文のステートメント／ループごとに遅延）

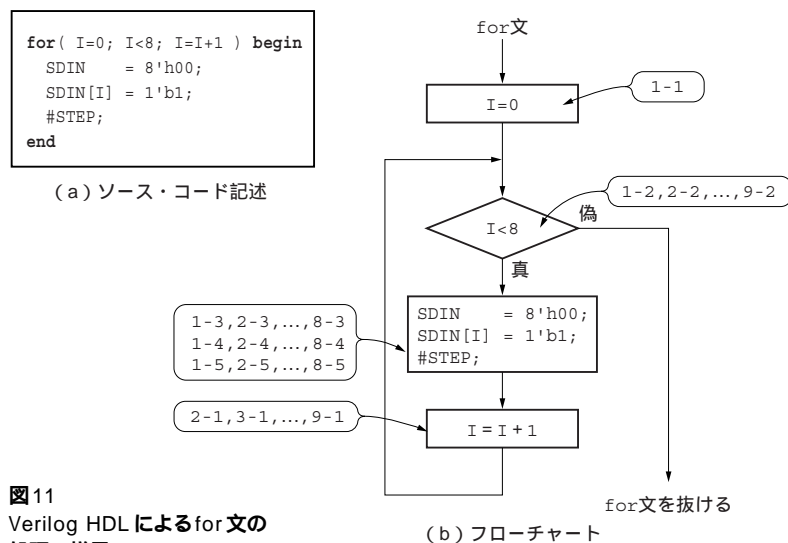
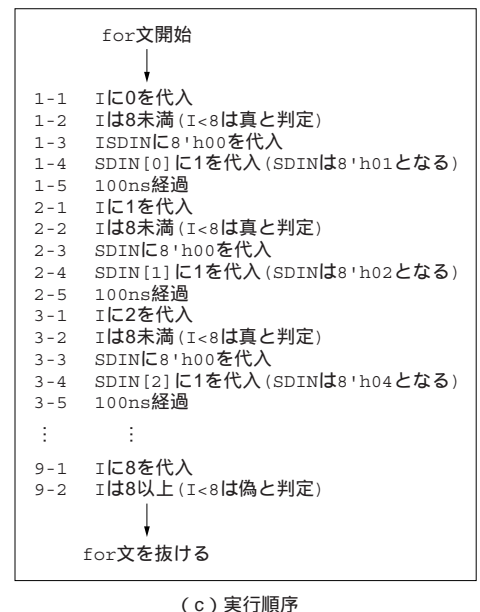



図11 Verilog HDLによるfor文の処理の様子
リスト5とまったく同じタイミングになることが分かる．

(c) 実行順序

リスト7　VHDLによるエンコーダ回路向けテストベンチ

```
library IEEE;
use IEEE.std_logic_1164.all;

entity encode_tb is
end encode_tb;

architecture SIM of encode_tb is

constant STEP: time := 100 ns;

component encode
   port (DIN  : in  std_logic_vector(7 downto 0);
         DOUT : out std_logic_vector(2 downto 0));
end component;

  signal SDIN  : std_logic_vector(7 downto 0);
  signal SDOUT : std_logic_vector(2 downto 0);

begin

uencode : encode port map (
  DIN     => SDIN,
  DOUT    => SDOUT);

process begin
  -- DINの各ビットに順次1を印加
  SDIN <= "00000001"; wait for STEP;
  SDIN <= "00000010"; wait for STEP;
  SDIN <= "00000100"; wait for STEP;
  SDIN <= "00001000"; wait for STEP;
  SDIN <= "00010000"; wait for STEP;
  SDIN <= "00100000"; wait for STEP;
  SDIN <= "01000000"; wait for STEP;
  SDIN <= "10000000"; wait for STEP;
  assert false;
end process;

end SIM;

configuration cfg_encode_tb of encode_tb is
  for SIM
  end for;
end cfg_encode_tb;
```

図12　VHDLによるfor文の書式と動作
VHDLのループ文である．

整数型の変数によるfor文の書式と動作を**図12**に示します．for文の変数はfor文の外で宣言する必要はありません．また，for文自体の加算を除いて，代入の記述を書くことはできません．

リスト8は，for文を使って**リスト7**とまったく同じタイミングのテスト入力を書いたものです．for文による処理の様子を**図13**に示します．

リスト8　VHDLによるfor文を使ったテストベンチ

```
constant STEP: time := 100 ns;

...

  signal SDIN  : std_logic_vector(7 downto 0);
  signal SDOUT : std_logic_vector(2 downto 0);

begin

uencode : encode port map (
  DIN     => SDIN,
  DOUT    => SDOUT);

process begin
  -- DINの各ビットに順次1を印加
  for I in 0 to 7 loop
    SDIN    <= (others=>'0');
    SDIN(I) <= '1';
    wait for STEP;
  end loop;
  assert false;
end process;
```

・未宣言で使用できる
・代入できない
・レンジで記述されたデータ・タイプになる（この場合はinteger）

ループごとに遅延

Verilog HDL　VHDL

for文を使う場合と使わない場合を比べると，for文を使う方がだいぶすっきりしているのではないでしょうか．記述する量が減ることには，作業量が減ること以上に人の手によるミスを減らす効果があります．テスト入力として同じように変化する信号を記述する場合でも，なるべく記述量の少ない，効率の良い記述を心がけましょう．

3．オーバフロー対策付き加算回路の検証

ここでは，**図14**のような回路を検証します．この回路は4ビットの入力ポートIN1，IN2から入力されたデータを加算して，その結果を出力ポートCSUMから出力する組み合わせ回路です．ただし，加算結果が16以上になった場合には，15を出力します．

この回路に対して，すべての値の組み合わせを与えて検証したい場合，入力値を書き並べる手法を取るととても大変です．しかし，for文を2重に使うことで，非常に簡単

(c) 動作順序

図13
VHDLによるfor文の処理の様子
リスト7とまったく同じタイミングになることが分かる．

(a) ブロック図

IN1 ＋ IN2	CSUM
15以下	IN1 ＋ IN2
16以上	15

IN1 ＋ IN2は真理値表の表記を簡略化したもの
IN1＝7，IN2＝7のとき，IN1 ＋ IN2は14なので
15以下，CSUMの値はIN1 ＋ IN2，つまり14
IN1＝9，IN2＝11のとき，IN1 ＋ IN2は20なので
16以上，CSUMの値は15

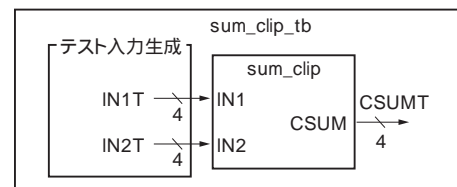
(b) 真理値表

(c) テストベンチの構成

図14 オーバフロー対策付き加算回路
4ビットの入力ポートIN1，IN2から入力されたデータを加算して，その結果を出力ポートCSUMから出力する組み合わせ回路である．

に書けるようになります(ただし，波形を目視で確認するのは大変)．テストベンチの構成を**図14(c)**に示します．テストベンチの階層にある信号名は，検証対象の回路のポート名と見分けがつきやすいように，最後にTを付けています．

Verilog HDL

リスト9に，for文を2重に使った記述例を示します．このテストベンチは**図15**のフローチャートのように動作します．

テスト入力の記述の一番外側にinitial文があります．initial文の中には，for文が一つと$finishの文が一つあるので，この二つのステートメントをbegin～endで囲ってあります．

initial文の直下のfor文(Iでループさせるfor文)の中には，for文が一つしかありません．ステートメントが一つなので，begin～endで囲う必要はありません(囲ってもかまわない)．

一番内側のfor文(Jでループさせるfor文)の中には，IN1T，IN2Tの代入文と#の文があります．ステートメントが三つなので，begin～endで囲う必要があります．

VHDL

リスト10に，for文を2重に使った記述例を示します．このテストベンチは**図16**のフローチャートのように動作します．

ここでは新たにconv_std_logic_vectorという関数が使われています．conv_std_logic_vectorは，整数など

リスト9　Verilog HDLによるオーバフロー対策付き加算回路のテストベンチ

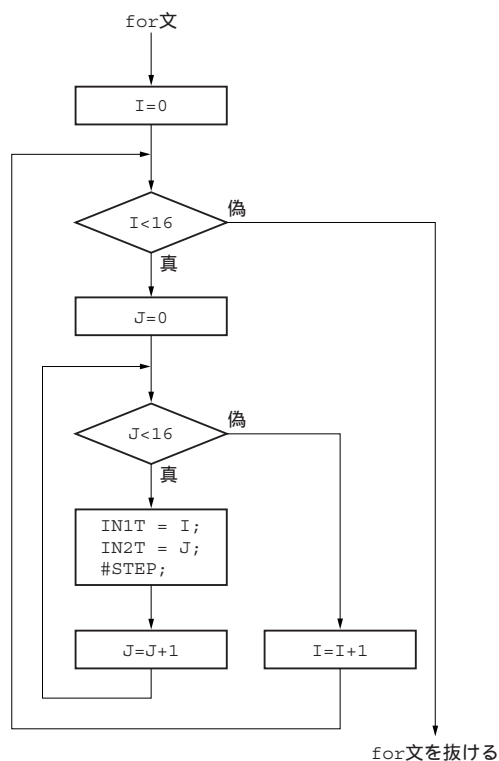
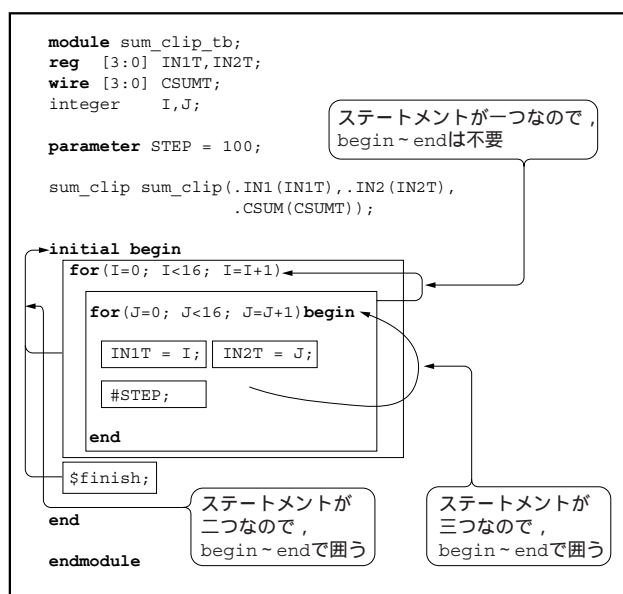
```
module sum_clip_tb;
reg  [3:0] IN1T,IN2T;
wire [3:0] CSUMT;
integer    I,J;

parameter STEP = 100;

sum_clip sum_clip(.IN1(IN1T),.IN2(IN2T),
                  .CSUM(CSUMT));

initial begin
  for(I=0; I<16; I=I+1)
    for(J=0; J<16; J=J+1)begin
      IN1T = I; IN2T = J;
      #STEP;
    end
  $finish;
end

endmodule
```


図15　Verilog HDLによる二重のfor文の動作
リスト9の動作をフローチャートで示す.

をstd_logic_vector型に変換する関数です．VHDLでは異なる型の変数(信号)同士で代入を行うことができないので，この関数を使い，整数型の変数I，Jを4ビットの

リスト10　VHDLによるオーバフロー対策付き加算回路のテストベンチ

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity sum_clip_tb is
end sum_clip_tb;

architecture SIM of sum_clip_tb is

constant STEP: time := 100 ns;

component sum_clip
   port (IN1,IN2 : in  std_logic_vector(3 downto 0);
         CSUM    : out std_logic_vector(3 downto 0));
end component;

  signal IN1T,IN2T : std_logic_vector(3 downto 0);
  signal CSUMT     : std_logic_vector(3 downto 0);

begin

usum_clip : sum_clip port map (
  IN1     => IN1T,
  IN2     => IN2T,
  CSUM    => CSUMT);

process begin
  for I in 0 to 15 loop
    for J in 0 to 15 loop
      IN1T <= conv_std_logic_vector(I, 4);
      IN2T <= conv_std_logic_vector(J, 4);
      wait for STEP;
    end loop;
  end loop;
  assert false;
end process;

end SIM;

configuration cfg_sum_clip_tb of sum_clip_tb is
  for SIM
  end for;
end cfg_sum_clip_tb;
```

(注記：conv_std_logic_vector(,)を使うために必要)

std_logic_vector型の信号に変換しています．

図17にconv_std_logic_vectorの書式を示します．変数名の部分に書かれた変数の値をビット幅の部分に書かれたビット幅で，std_logic_vector型に変換して，信号名の部分に書かれた信号に代入します．なお，conv_std_logic_vectorを使うためには，パッケージstd_logic_arithが必要です．

Verilog HDL　VHDL

図18に**リスト9**と**リスト10**のタイミング・チャートを示します(CSUMは期待値)．

4. まとめ

テストベンチではHDLのすべての文法を使うことができます．

今回解説した遅延の記述とfor文のように，まったく同

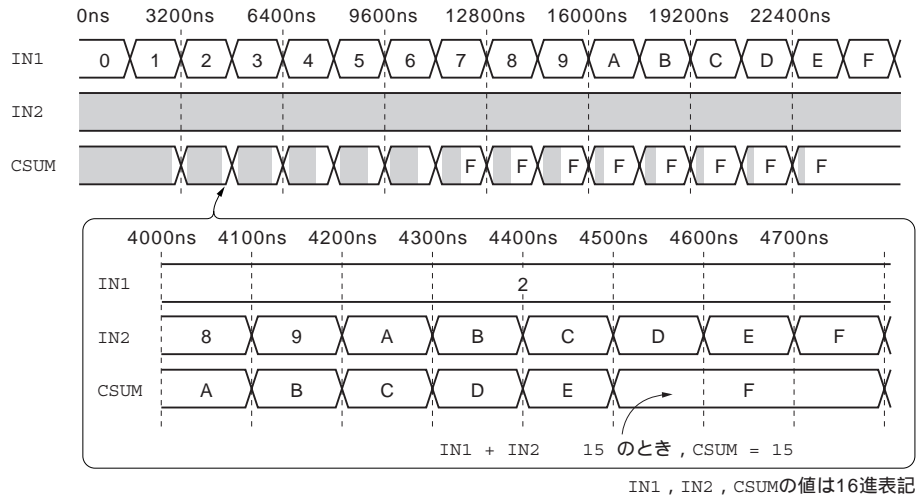



図18
オーバフロー対策付き加算回路のテストベンチのタイミング・チャート
リスト9とリスト10の動作を示す．

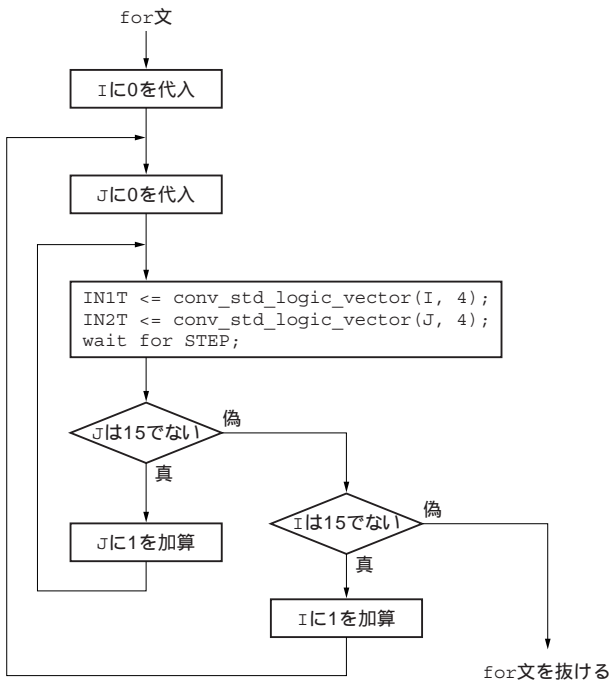



図16　VHDLによる二重のfor文の動作
リスト10の動作をフローチャートで示す．

```
<信号名> <= conv_std_logic_vector( <変数名> , <ビット幅> )
```

図17　conv_std_logic_vectorの書式
変数名の部分に書かれた変数の値をビット幅の部分に書かれたビット幅で，std_logic_vector型に変換して，信号名の部分に書かれた信号に代入する．パッケージstd_logic_arithが必要．

じ機能を果たす記述が複数通りある場合には，使いやすく，記述量が少なくて，間違いの起こりにくい記述を選ぶよう心がけてください．

ところで，**図18**のタイミング・チャートのように，テスト・パターンが長くなると，波形を目視で確認するのには非常に時間がかかるようになります．波形の目視は，バグや配線ミスを大量に含む検証の初期段階では，直感的にバグを発見しやすいので有効な方法です．しかし，すでにいくつかのバグが修正された検証の中盤以降では，新たなバグを修正したつもりが，その修正前に正常に機能していた回路に，別のバグを追加してしまうことがあります．このような状況では，波形の目視だけではバグを見逃しやすくなってしまうので，波形目視以外の方法を考えなくてはいけません．

HDLによる回路の検証手段としては，波形目視以外に，標準出力による確認やファイル出力による確認，テストベンチの中での比較による確認といった方法があります．次回は，新しい確認手段として，標準出力の文法を解説します．

やすおか・たかし
(株)エッチ・ディー・ラボ

＜筆者プロフィール＞
安岡 貴志．東京理科大学 理工学部 数学科卒業．前職のデザインセンターでは，3年間Verilog HDLによるASIC開発に携わる．2002年にエッチ・ディー・ラボに入社し，Verilog HDL，VHDL，SystemCによる開発に従事するほか，同社のトレーニング講師を務める．